AR-006-481

ELECTRONICS RESEARCH LABORATORY

Communications Division

GENERAL DOCUMENT
ERL-0544-GD

A GENERAL LEAST-SQUARES FITTING PACKAGE FOR THE SUN WORKSTATION

by

S. C. Troedson and A. C. Lindsay

SUMMARY

This document describes a general least-squares fitting software package, featuring an easy to use and flexible user interface. It is implemented on a Sun workstation and designed for the public domain.

APR 91

COPY No.

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory,PO Box 1600, Salisbury, South Australia, 5108

CONTENTS

FIGURES

APPENDICES

1 INTRODUCTION

This document introduces a user interface that provides fast and efficient general least-squares analysis of experimental data. It is suitable for use in the Sunwindows environment on a Sun UNIX workstation. The package is capable of providing solutions to both linear and non-linear least-squares models of experimental data, and incorporates full graphics and error analysis. Known uncertainties in the data are easily incorporated as "weights" for each experimental point if required.

The routine is user specific and as such a limited knowledge of UNIX as well as C and/or Pascal is necessary. However an attempt has been made to lessen this burden as much as possible without destroying the flexibility of the program. An overview of the operational details of the package is given in Section 2. Examples of how to use the package are given in Section 3.

The linear least-squares analysis allows unweighted, y-weighted and x- and y-weighted optimisation, and in the former cases either the slope or intercept may be fixed if required. Since the theory of unweighted and y-weighted linear least-squares is well known[1],[2], it will not be described in this document. The theory for the case where both the dependent and independent variables exhibit uncertainties is not well known however, and as such is outlined in Appendix A.

Appendix B details the theory of non-linear least-squares optimisation using the Levenberg-Marquardt algorithm, which is the routine implemented in this package. The routine incorporates options for unweighted or y-weighted fitting of models with multiple parameters (maximum specified by the user), any of which may be easily fixed or floated as desired. An outline of the method used to estimate the errors in the final parameters is also given.

The package is public domain software and as such there is no guarantee it is bug free. It may be freely distributed and augmented, the only request being that the original and subsequent substantial authors' names remain at the top of the major routines.

2 OPERATIONAL DETAILS

In this section a description of the system and the details of establishing an appropriate environment are given. Figure 1 shows an overview of the environment in which **spg_fit** operates. The role of each of the modules is discussed below.

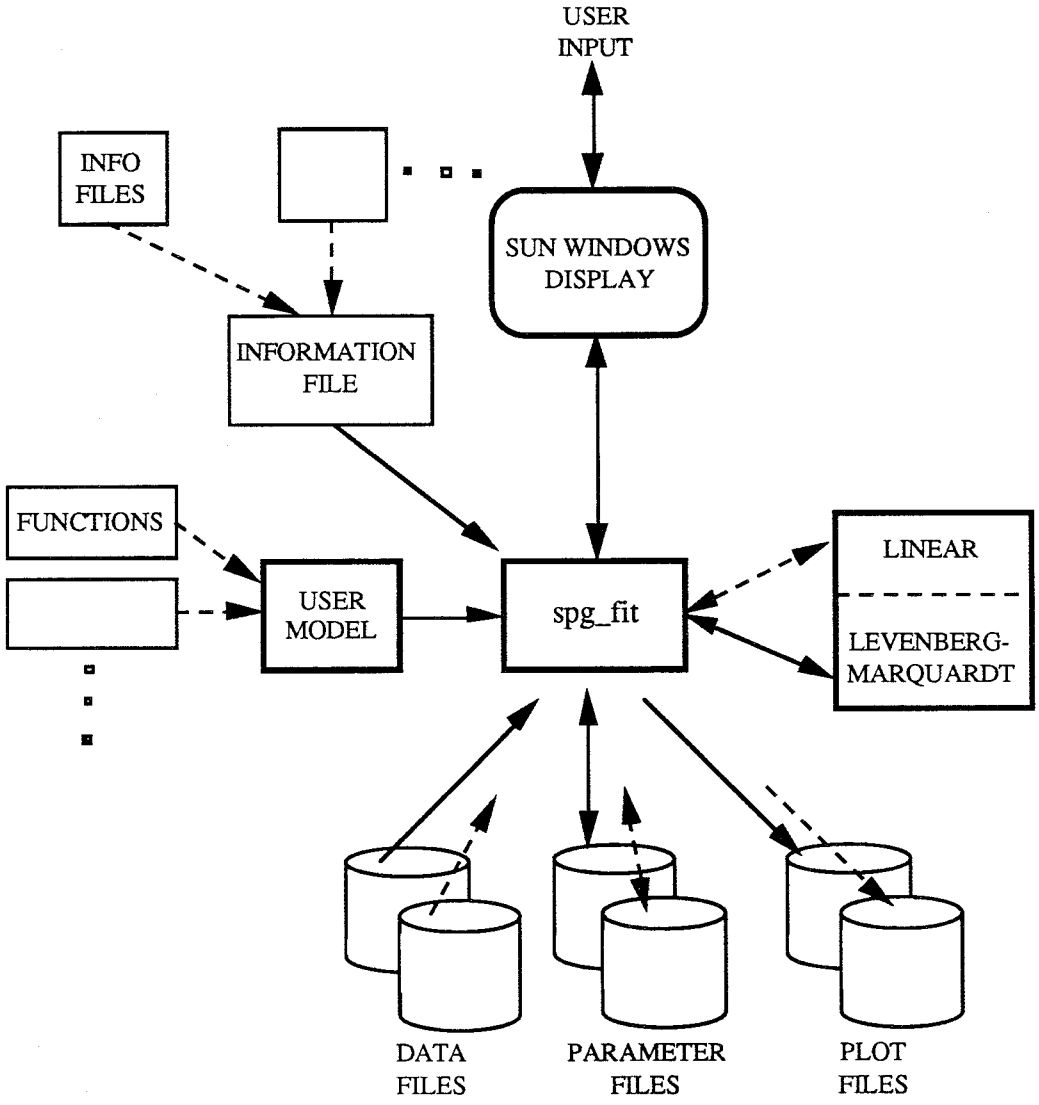

Figure 1 The file environment of spg_fit

2.1 The function file

The function file supplies both the model and, optionally, the derivatives of the model with respect to each parameter. The function may be written in either C or Pascal. Note however, that if the routine is written in Pascal, all call parameter arrays must be **var** since the main **spg_fit** C routine communicates using pointers. Details of the call parameters may be found in **README** included on the archive and given in Appendix C. The filenames must be either **cfn.c** or **pfn.p** for the C and Pascal routines respectively.

Examples of both C and Pascal routines are included on the tape (in the directory **Fit_lib**) and are given in Appendix D. They include simple models such as trigonometric, exponential and distribution functions. The user must copy the required routine into the appropriate file (**cfn.c or pfn.p**) and recompile (see Section 2.7). If the **spg_fit** routine is recompiled without the presence of the function file in the current working directory, a dummy function will be automatically linked. In this case only the linear fitting mode will be operational (see Section 2.6.2).

As noted, supplying the first derivatives of the function with respect to each parameter is optional. If the analytic derivatives are supplied, the program will run slightly faster and the usual round-off errors associated with numerical evaluation of the derivatives will be avoided. However, if the derivatives are not supplied in the function file, **spg_fit** will automatically recognise this and numerical evaluation of the derivatives will be undertaken.

2.2 The data file

The data file is a text file containing the experimental data and optionally any experimental variances. The format of the data file is simply:

x_1	y_1	$\sigma^2_{x_1}$	$\sigma^2_{y_1}$
x_2	y_2	$\sigma^2_{x_2}$	$\sigma^2_{y_2}$
.	.	.	.
.	.	.	.
.	.	.	.
x_n	y_n	$\sigma^2_{x_n}$	$\sigma^2_{y_n}$

If there are only three columns, the third column is automatically interpreted as σ^2_y's, and if there are only two columns the program recognises that the uncertainties of the individual data points are unknown. The columns may be separated by either spaces or tabs. For graphics purposes, the data {x} should be presorted into either ascending or descending order.

2.3 The function information file

The function information file, called **fn_info**, is a text file that contains the names of the model parameters, entered line by line. The routine uses the text strings from this file for display purposes and also to calculate the number of parameters in the model. It is only required for the non-linear mode of operation. It is important that the order of the parameter names be the same as the order in which the parameters are specified in the user's routine. Similarly, it is important to remember to change the **fn_info** file whenever the non-linear function is changed.

2.4 The curve save file

The curve save file is named and established interactively from the main control panel and contains the theoretical curve whenever "Save" is pressed. Saving the curve is *not* done automatically.

2.5 The parameter save file

The parameter save file is again named and established interactively from the main control panel. This feature is useful not only for storing the final analysis results but can be used to provide an initial guess for parameter values based on a previous fit. This is achieved by specifying the appropriate parameter save file on the main control panel and clicking the "Load" button. When "Save" is clicked, the specified file will contain the parameters as well as their corresponding errors.

2.6 The interface

The interface consists of a main control window and display canvas, a "linear fit" panel and a "non-linear fit" panel. The "fit" panels are pop-up windows selected by choosing either the "linear" or "non-linear" options on the main control panel via the "select model type" toggle. The function and layout of each of the three main panels is described in the following sections.

2.6.1 The main control window and display canvas

The main control window and display canvas appear immediately the programme is executed. Figure 2 shows the layout of this window along with the linear fit options panel.

The field **Directory** contains the current working directory which may be changed at any time. Files for data and saving may be entered into the indicated fields. The toggle shown in the top centre allows switching between the non-linear and the linear modes. Figure 3 shows the non-linear selection.

The display graphics has two modes of operation:

- If a datafile is specified, the routine will calculate ranges and display the data points with any associated error bars (the error bar length is defaulted to two standard deviations either side of the point). Once a parameter has been specified a curve will be drawn spanning the data.

- Alternatively if no data is specified but ranges and parameters are correctly input, the routine will display the model curve over the selected horizontal range of the graph on the display canvas. This is useful when examining the behaviour of the model.

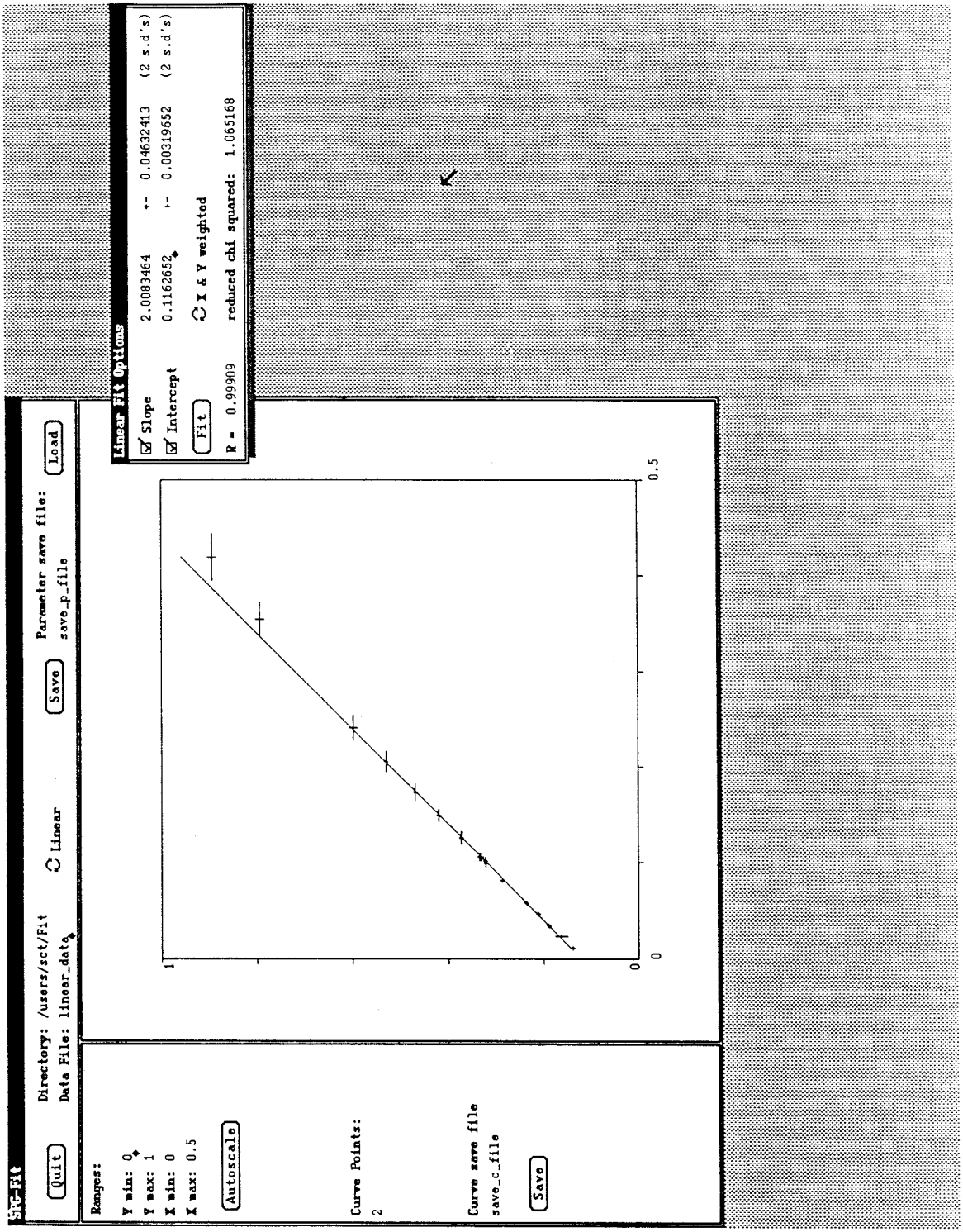

Figure 2 The linear mode of operation

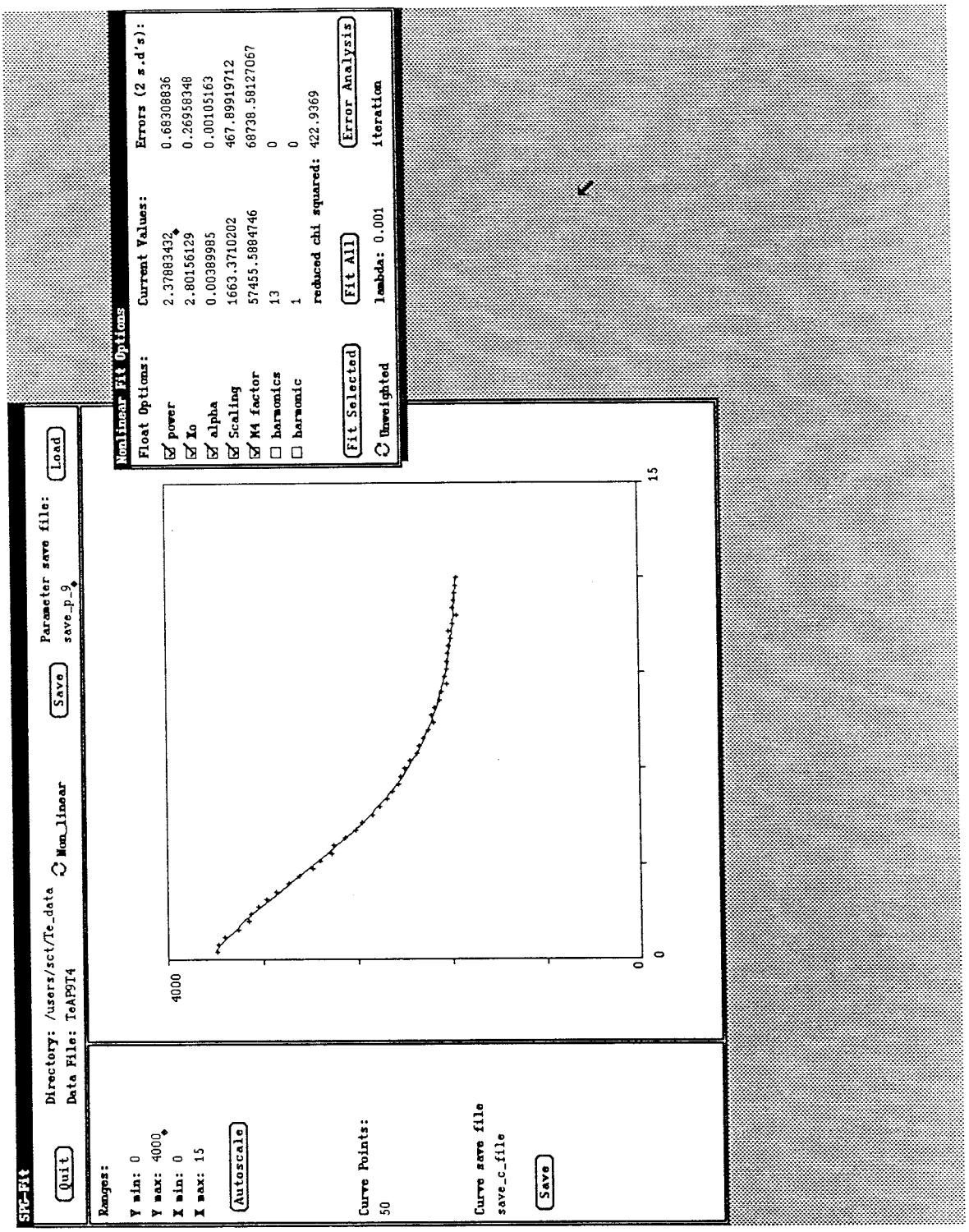

Figure 3 The non-linear mode of operation

The **Curve Points** prompt on the side panel refers to the number of points the routine will use to draw the curve. The default is set to 100, or, when there is data, to the number of input data points. Problems with aliasing will only affect the display and not the fitting. During the optimisation this number will be equal to the number of data points, since **spg_fit** will obtain the values of these points from the Levenberg-Marquardt call. Upon completion the number of curve points will revert to that specified.

2.6.2 The linear fit panel

The linear fit panel will appear upon selection of the "linear fit" option from the toggle on the top control panel.

Figure 2 shows the linear least-squares mode of operation. Beside each parameter is a box which may be checked or left clear if floating or fixing of the parameters is required. A weighted correlation coefficient (R) is calculated and displayed, as well as a value for the (reduced) weighted sum of the square of the residuals (**reduced chi squared**). The toggle has three choices, unweighted, y-weighted and x- and y-weighted. Only in the latter case is the option to fix parameters not included. By fixing the slope at zero, this panel may be used to calculate either the weighted or unweighted mean of the data, along with the associated uncertainty.

2.6.3 The non-linear fit panel

The non-linear fit panel will appear upon selection of the "non-linear fit" option from the toggle on the main control panel. This is shown in Figure 3. This mode of operation may be used when the model is non-linear with respect to the parameters or for generalised linear least-squares analysis.

The layout and operation is similar to the linear case. The values of the parameters may be changed at any time, with the routine updating the screen automatically. The χ^2 value corresponding to the new parameter value will also be displayed if the **Curve Points** (as described in Section 2.6.1) is equal to the number of data points. This is so that two function calls are not required, ie., one for the plot and one for the χ^2 calculation, as this would be time consuming for complex functions. If **Curve Points** is not equal to the number of data points, then the χ^2 value is only updated once the non-linear fitting is in progress.

Once the floating parameters are chosen and the weighting option selected, "Fit Selected" may be chosen to begin the optimisation. "Fit All" will select all parameters to float. If, after an iteration, the fit is found to have improved, the routine will update the parameters on the screen, including the Levenberg Marquardt parameter (**lambda**) and the value for χ^2. By pressing the L1 (Stop) key on the Sun keyboard the fit may be halted at any stage.

This is not immediate since the routine waits for any internal user function calls to be completed. The parameter lambda may be specified before fitting to optimise the convergence, however this should rarely be necessary.

Once optimisation is completed the "Error Analysis" button may be chosen to calculate the uncertainty in each parameter. This is taken as two standard deviations (95%), however this may be altered by changing the appropriate field in "**spg_fit.h**".

2.7 Setting up the spg_fit environment

The user must execute a small number of steps in order to customise the routine for his/her purposes. They are:

1. Create the source file (**cfn.c** or **pfn.p**), discussed in Section 2.1, containing the non-linear model. This is not necessary when a linear least squares fit is all that is required.

2. Create the corresponding **fn_info** discussed in Section 2.3. If this does not exist the routine will continue, however only the linear least squares mode will be operational.

3. Customise the compile parameters. These parameters are found in the files **Makefile** and **spg_fit.h** (see Appendix C) and should be checked before compiling.

4 Re-compile using a simple **make** command.

Details of all these steps may be found in the **README** file supplied on the archive, and in Appendix C.

2.8 The algorithms

It is the intention of this package to provide a convenient means of undertaking the task of general least-squares modelling in as simple a manner as possible for the user. It is also the intention of the authors that the code be accessible to the user if changes or updates are considered necessary. It is useful therefore to have some understanding as to how the optimisation routines used in **spg_fit** actually work. To this end, in order to make this document as self-contained as possible, Appendices A and B contain a basic outline of:

(i) linear least-squares optimisation for the case when both the dependent *and* the independent variables are subject to some uncertainty,

(ii) the theoretical basis of the Levenberg-Marquardt non-linear optimisation algorithm used in **spg_fit**, and

(iii) a discussion of the estimation of the statistical errors associated with the optimised parameters for the case of non-linear functions.

As noted in the introduction, the common cases of unweighted linear least-squares fitting and of y-weighted linear least-squares fitting are well known, and as such will not be described in this report. Those unfamiliar with the basis of these algorithms are referred to references [1] and [2], or to any of the standard undergraduate mathematical texts dealing with numerical techniques.

3 AN EXAMPLE - FITTING A POWER FUNCTION ($y=A+B*x^n$)

In this section an example is followed through to demonstrate the procedures of using **spg_fit**.

The first step is to copy the user function and information files into the appropriate filenames. For this example:

cp /source_path/Fit_lib/fit_power_fn.c cfn.c
cp /source_path/Fit_lib/info_power_fn fn_info

where **source_path** is the **spg_fit** source directory.

Now the routine is ready for compiling. This may be done from any directory which includes the above two files. To compile from a directory other than the **spg_fit** source type

make -f /source_path/Makefile fitc

If you are working from the source directory then simply type

make fitc

(**fitc** is used because in this example the function is written in C).

This process will leave the **spg_fit** object files in the source directory and place the function object file in the current directory. The executable will reside in the path specified by **DEST** which is defined in the Makefile or overridden on the command line.

The routine is now executed by typing **fit** at which time the main control window appears. By pressing over the top centre toggle (labelled "**Select Model Type**") with the left mouse button, a non-linear options panel will appear as shown in Figure 4. From this point the user may examine the properties of the function or input a datafile in the appropriate field.

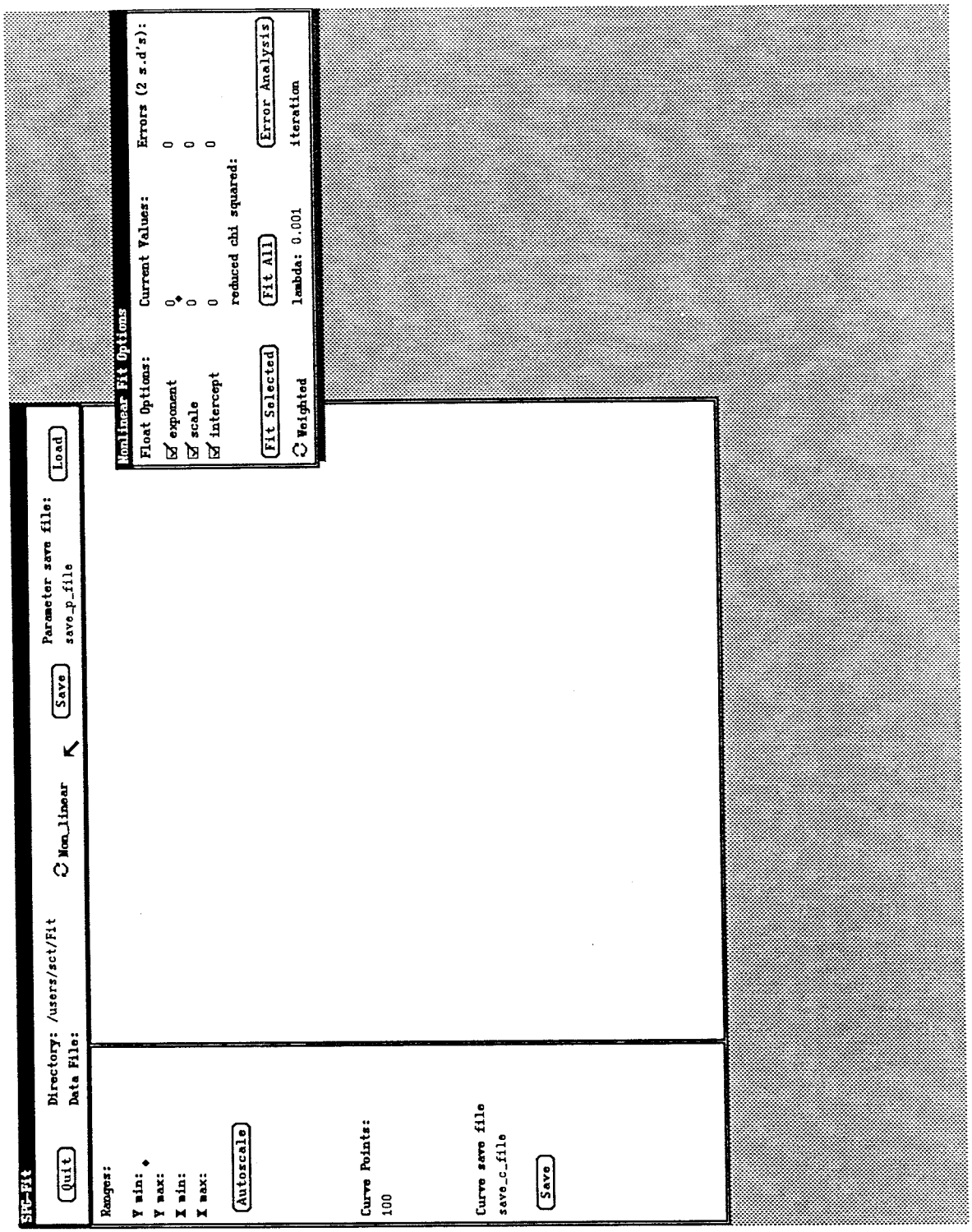

Figure 4 The initial screen display

In Figure 5 the file "power_noisy" has been entered and displayed. (No error bars for this data). Note that the number of samples is automatically set to the number of data points. At this point parameter values may be changed manually to experiment with the data, as shown in Figure 6.

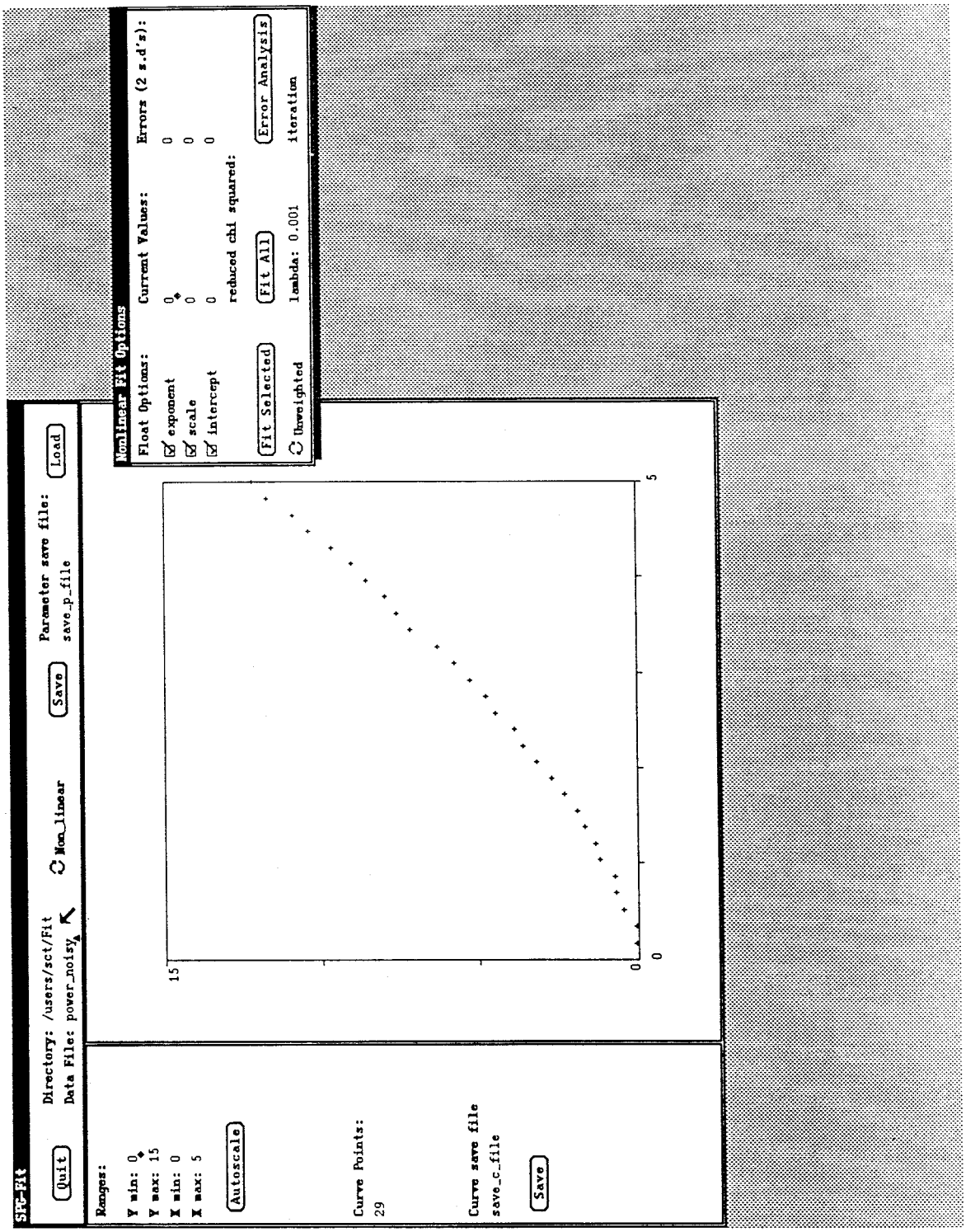

Figure 5 Sample Data

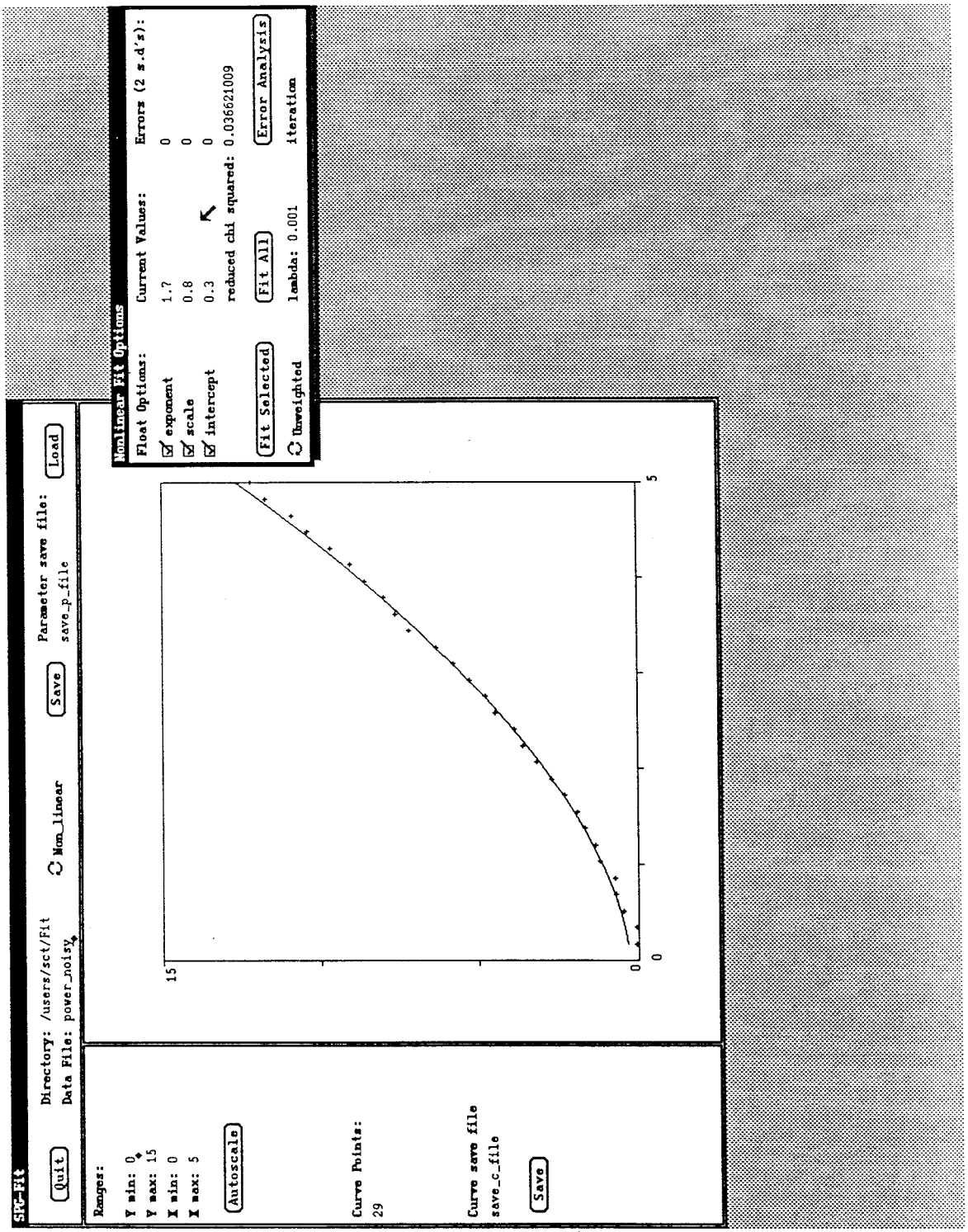

Figure 6 Experimenting with the parameters

It is initially desired to fit "**power_noisy**" with all the parameters, hence either all boxes are ticked and "**Fit Selected**" chosen , or simply the button "**Fit All**" is selected. With some functions the routine may fail if some of the parameters are set to zero at the start. In the power function for instance, setting "scale" to zero and fitting will return IEEE NaN's. By starting the optimisation with the parameter **scale** = 0.1 and performing an error analysis produces the screen shown in Figure 7. The data in this example was generated with the function $y=x^{1.56}$ with additive gaussian noise of variance 0.1. Note the small value of **lambda** at the completion of the fit. Figure 8 demonstrates a fit when one of the parameters is required to be fixed, in this case the intercept. A zero error is returned for the fixed parameter. Once the fit is satisfactory the number of samples may be increased and the curve saved so that a quality graphics package may be used to obtain a printout.

The power function example is particularly interesting in that it also demonstrates a pitfall that is occasionally encountered when using the Levenberg-Marquardt algorithm. If the optimised parameters shown in Figure 7 are changed very slightly - say in the last one or two digits - and the fit re-started, very often the Marquardt parameter will be seen to "explode", there being however little or no associated change in the parameter values. This is due to the fact that the routine can have a tendency to wander around the minimum if the χ^2 hypersurface exhibits a generally flat valley with complicated topology in the region of the optimal parameter values[2]. In this case a very small change in the parameter values can produce a large correction, which will be rejected and the value of λ subsequently increased. This procedure can occasionally "run away". This is one example where the parameter values are valid despite an associated large value for the Marquardt parameter.

Due to the possibility of the Marquardt parameter "running away", a flexible ceiling has been placed on the value of the maximum number of iterations (see **MAX_ITERATIONS** in **spg_fit.h**). The problem may be overcome by choosing different starting values, or changing the starting value of λ.

Another problem the user may encounter is the convergence of the parameters to a local minimum. This routine does not supply any algorithms such as Monte Carlo searching to address this problem. Hence there must be some judgement in choosing appropriate starting values for the parameters if there are multiple minima in the hypersurface. If there are no *a priori* reasons for choosing starting values for a particular parameter then a manual search for other minima could be undertaken, taking note of the χ^2 values in turn.

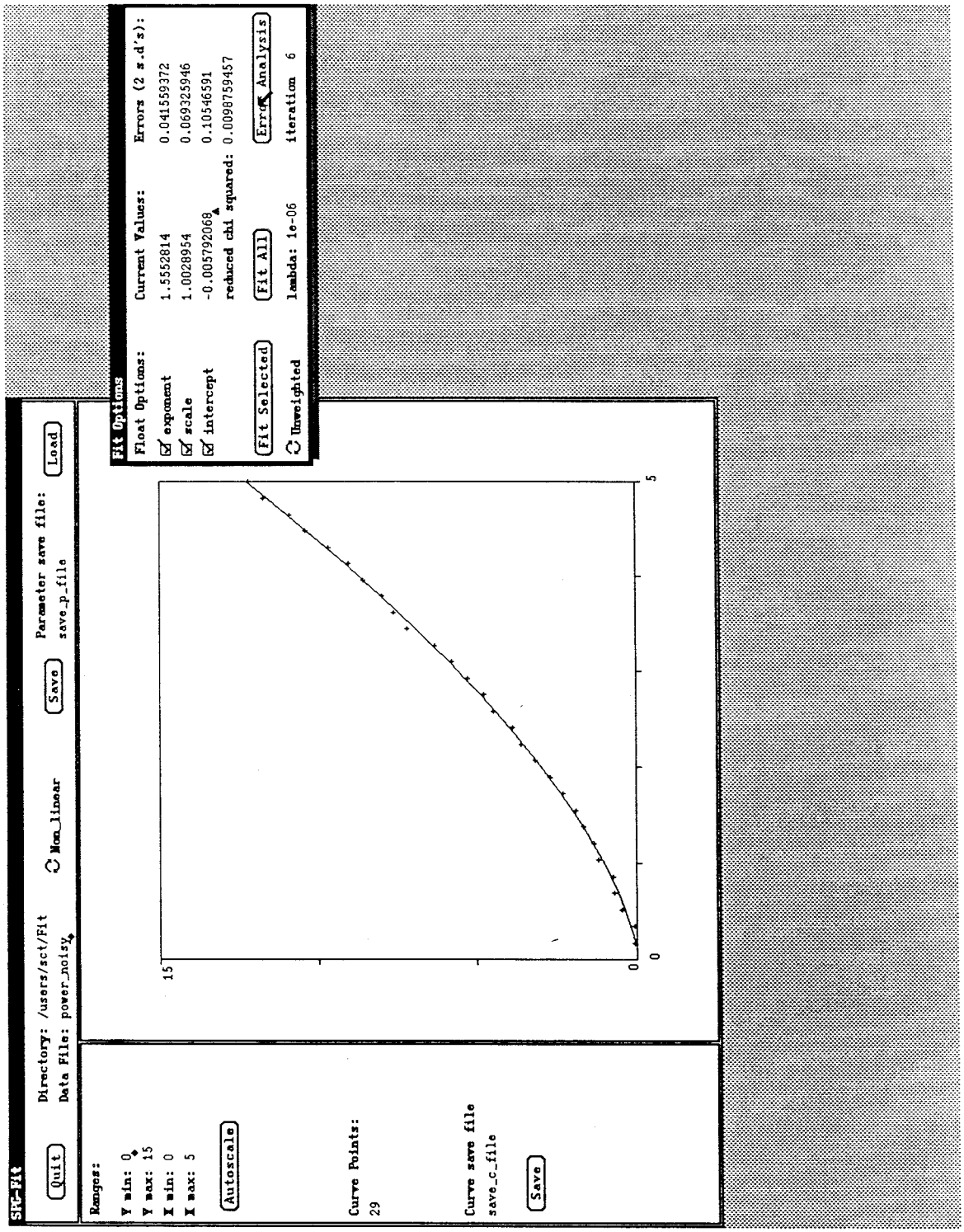

Figure 7 Completed fit with error analysis

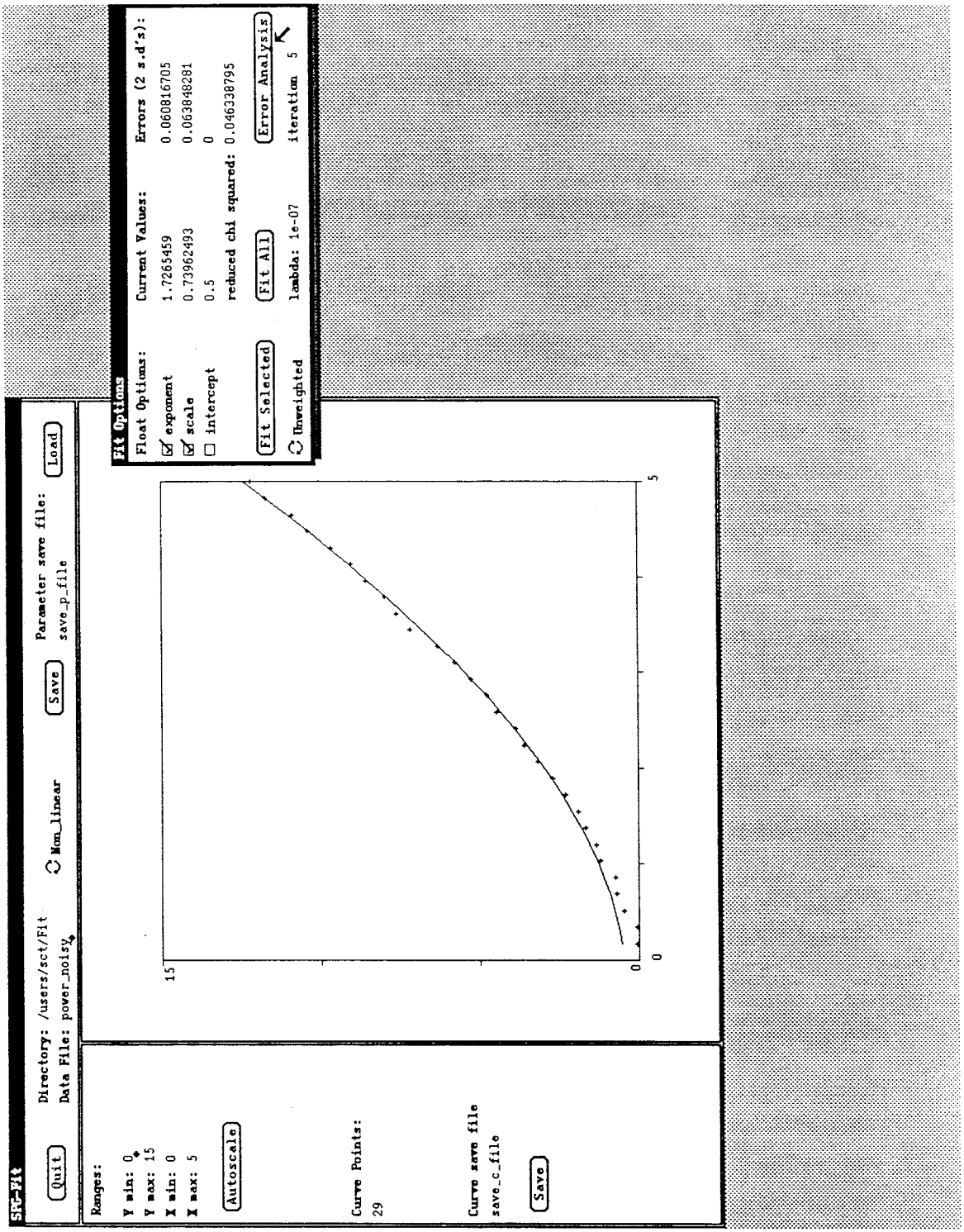

Figure 8 A fit with intercept fixed

REFERENCES

1	Bevington, P.R.,	*Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill Book Company, New York, 1969.
2	Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T.,	*Numerical Recipes*, The Art of Scientific Computing, Cambridge University Press, Cambridge, 1986.
3	York, D.,	"Least-squares Fitting of a Straight Line", Can. J. Phys., 44, 1079-1086, 1966.

APPENDIX A

LINEAR LEAST-SQUARES FITTING TO DATA EXHIBITING UNCERTAINTIES IN BOTH THE DEPENDENT AND INDEPENDENT VARIABLES

CONTENTS

LINEAR LEAST-SQUARES FITTING TO DATA EXHIBITING UNCERTAINTIES IN BOTH THE DEPENDENT AND INDEPENDENT VARIABLES

A.1 Introduction

The solution to the problem of finding the best linear least-squares fit to data which exhibits uncertainty in the dependent variable is well known, and will not be considered here. The more interesting, and probably more realistic case of finding the "best" description of data when there exists uncertainty in both the dependent and independent variables is the subject of this section.

The case where both variables exhibit some uncertainty has generally led to the use of some approximation method to determine the best fit to the data. For example, one may associate all of the error with the y co-ordinate, calculate the usual weighted least-squares solution and then do a similar calculation with all the error associated with the x co-ordinate. Since the solutions obtained for each case will in general be different, the arithmetic mean is taken as an approximate solution. There are of course more sophisticated methods, however the most general solution appears to be that derived by York[3]. The York algorithm is used in **spg_fit**, and is derived in the next section.

A.2 The York algorithm

Let x_i and y_i be the actual data points with X_i and Y_i denoting the fit estimates, i.e.

$$Y_i = a + bX_i \tag{A.1}$$

The minimising function will be taken as

$$S = \sum_i \frac{1}{\sigma_{x_i}^2}(X_i - x_i)^2 + \frac{1}{\sigma_{y_i}^2}(Y_i - y_i)^2 \tag{A.2}$$

where the $\sigma_{x_i}^2$ and $\sigma_{y_i}^2$ are the variances for each point, which may be determined during data acquisition. If the variances are not known, they are automatically set to unity by **spg_fit**.

Following the usual procedures, minimising equations (A.1) and (A.2) implies the conditions

$$\delta S = \sum_i \frac{1}{\sigma_{x_i}^2}(X_i - x_i)\delta X_i + \frac{1}{\sigma_{y_i}^2}(Y_i - y_i)\delta Y_i = 0 \tag{A.3}$$

$$-\delta Y_i + \delta a + b\delta X_i + X_i\delta b = 0 \tag{A.4}$$

York then multiplies each of equations (i=1,...) (A.4) by its own Lagrangian undetermined multiplier λ_i and takes the sum of the ensuing equations plus equation (A.3), giving

$$\sum_i [b\lambda_i + \frac{1}{\sigma_{x_i}^2}(X_i - x_i)]\,\delta X_i + \sum_i [\lambda_i + \frac{1}{\sigma_{y_i}^2}(Y_i - y_i)]\,\delta Y_i + \delta a \sum_i \lambda_i + \delta b \sum_i \lambda_i X_i = 0 \qquad \text{(A.5)}$$

which, by equating the coefficients to zero, implies the conditions

$$X_i = -\lambda_i b \sigma_{x_i}^2 + x_i \qquad \text{(A.6a)}$$

$$Y_i = \lambda_i \sigma_{y_i}^2 + y_i \qquad \text{(A.6b)}$$

$$\sum_i \lambda_i = 0 \qquad \text{(A.6c)}$$

$$\sum_i \lambda_i X_i = 0 \qquad \text{(A.6d)}$$

Substituting equations (A.6a) and (A.6b) into equation (A.1) gives the expression for λ_i as

$$\lambda_i = W_i (a + bx_i - y_i) \qquad \text{(A.7)}$$

where

$$W_i \equiv \frac{1}{b^2\sigma_{x_i}^2 + \sigma_{y_i}^2} \qquad \text{(A.8)}$$

Substituting equation (A.7) into equation (A.6c) gives an expression for the intercept as

$$a = \frac{\sum_i W_i y_i - b \sum_i W_i x_i}{\sum_i W_i} \qquad \text{(A.9a)}$$

$$= \bar{Y} - b\bar{X} \qquad \text{(A.9b)}$$

where the "bar" terms are defined by

$$\bar{Z} \equiv \frac{\sum_i W_i Z_i}{\sum_i W_i} \qquad \text{(A.9c)}$$

i.e. the line will go through the "centre of gravity" of the data.

Introducing the quantities $U_i = x_i - \bar{X}$ and $V_i = y_i - \bar{Y}$, equation (A.9b) can be used to simplify equation (A.7) for λ_i to

$$\lambda_i = W_i (bU_i - V_i) \tag{A.10a}$$

and then equation (A.6a) can be expressed as

$$X_i = - W_i (bU_i - V_i) \sigma_{x_i}^2 + x_i \tag{A.10b}$$

Using these simplifications, equations (A.10) are substituted into equation (A.6d) and terms in powers of b are gathered. The result is

$$b^3 + 3\alpha b^2 + 3\beta b + \gamma = 0 \tag{A.11a}$$

where

$$\alpha \equiv - \frac{2\sum_i \sigma_{x_i}^2 W_i^2 U_i V_i}{3\sum_i \sigma_{x_i}^2 W_i^2 U_i^2} \tag{A.11b}$$

$$\beta \equiv \frac{\sum_i \sigma_{x_i}^2 W_i^2 V_i^2 - \sum_i U_i W_i x_i}{3\sum_i \sigma_{x_i}^2 W_i^2 U_i^2} \tag{A.11c}$$

$$\gamma \equiv \frac{\sum_i W_i V_i x_i}{\sum_i \sigma_{x_i}^2 W_i^2 U_i^2} \tag{A.11d}$$

Equation (A.11a) is called the "least-squares cubic" by York, although it isn't really a cubic, since b appears in the definition of W_i. The form of equations (A.11c) and (A.11d) is slightly different to the the definitions given in [3], however it can be shown that the expressions are equivalent. Equation (A.11a) can be solved as a cubic if a normal unweighted least-squares fit is performed and the estimate thus obtained for b used in the calculation of W_i. The required solution to equation (A.11a) is given by[2]

$$b_{new} = -2(\alpha^2 - \beta)^{1/2} \cos\left(\frac{\phi + 4\pi}{3}\right) - \alpha \qquad \text{(A.12a)}$$

where

$$\cos\phi = \frac{\alpha^3 - \frac{3}{2}\alpha\beta + \frac{1}{2}\gamma}{(\alpha^2 - \beta)^{3/2}} \qquad \text{(A.12b)}$$

Equation (A.12a) is the third root [3] of the least-squares "cubic", which always appears to be the required solution. By iterating this solution, the accuracy can be improved to any desired level. The convergence properties associated with the technique of treating equation (A.11a) as a pseudo-cubic and iteratively solving for b using equation (A.12a) have been checked for a number of sets of data by using a simple bisection search to solve for the appropriate root of equation (A.11a) directly. In the experience of the authors the pseudo-cubic technique has been reliable, and as such the technique is used in **spg_fit**. Once the optimal solution is achieved, the final value for b is then used one last time to calculate the corresponding intercept, a, from equation (A.9a).

Having obtained the optimum values for the slope and intercept, the residuals can be calculated from equations (A.6a), (A.6b), (A.7) and (A.10a) in one of several forms, i.e.

$$X_i - x_i = bW_i(bU_i - V_i)\,\sigma^2_{x_i} = bW_i(a + bx_i - y_i)\,\sigma^2_{x_i} \qquad \text{(A.13a)}$$

$$Y_i - y_i = W_i(bU_i - V_i)\,\sigma^2_{y_i} = W_i(a + bx_i - y_i)\,\sigma^2_{y_i} \qquad \text{(A.13b)}$$

which give an expression for the minimisation function S defined by equation (A.2) as

$$S = \sum_i \frac{1}{\sigma^2_{x_i}}(X_i - x_i)^2 + \frac{1}{\sigma^2_{y_i}}(Y_i - y_i)^2$$

$$= \sum_i W_i^2(bU_i - V_i)^2\left(b^2\sigma^2_{x_i} + \sigma^2_{y_i}\right)$$

$$= \sum_i W_i^2(a + bx_i - y_i)^2\left(b^2\sigma^2_{x_i} + \sigma^2_{y_i}\right) \qquad \text{(A.14)}$$

The program **spg_fit** displays the result of calculating equation (A.14) divided by the degrees of freedom, next to the message "**reduced chi squared:**" which appears in the "linear fit" panel during execution.

In order to estimate the errors in the fit parameters, York uses a Taylor's series expansion for the straight line about the assumed values for slope, intercept and estimated positions of the points[3]. The resulting expressions are:

$$\sigma_b^2 = \frac{1}{n-2} \frac{\sum_i W_i(bU_i - V_i)^2}{\sum_i W_i U_i^2} \qquad \text{(A.15a)}$$

$$\sigma_a^2 = \sigma_b^2 \frac{\sum_i W_i x_i^2}{\sum_i W_i} \qquad \text{(A.15b)}$$

The 95% confidence error bars are then given by $\delta a = 2\sigma_a$ and $\delta b = 2\sigma_b$ as usual.

APPENDIX B

LEAST-SQUARES MINIMISATION OF A NON-LINEAR FUNCTION

CONTENTS

LEAST-SQUARES MINIMISATION OF A NON-LINEAR FUNCTION

B.1 Introduction

In this section an outline of the development of the Levenberg-Marquardt algorithm for non-linear least-squares optimisation and the method of determining the error estimates for the fit parameters is presented.

B.2 The χ^2 hypersurface

Consider an arbitrary function y(x;a) defined at the points x_i (i=1..N) which depends in a non-linear manner on parameters a_j (j=1..M). The parameters **a** are to be chosen so that in some sense they model the set of experimental data points y_i that represent the function y(x;**a**), each of which has a measured standard deviation σ_i. In order to model the data, it is desirable to minimise the usual least-squares norm

$$\chi^2 \equiv \sum_{i=1}^{N} \frac{[y_i - y(x_i;\mathbf{a})]^2}{\sigma_i^2} \tag{B.1}$$

as a function of the parameters a, i.e

$$\frac{\partial \chi^2}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_{i=1}^{N} \frac{[y_i - y(x_i;\mathbf{a})]^2}{\sigma_i^2} = 0 \tag{B.2}$$

The quantity χ^2 describes an M-dimensional hypersurface which must be searched to find the appropriate minimum value of χ^2. Carrying out the differentiation yields

$$\frac{\partial \chi^2}{\partial a_j} = -2 \sum_{i=1}^{N} \frac{[y_i - y(x_i;\mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i;\mathbf{a})}{\partial a_j} \tag{B.3}$$

It will be useful later to have the matrix of second derivatives of the χ^2 hypersurface - the *Hessian matrix*. This is given by

$$\frac{\partial^2 \chi^2}{\partial a_j \partial a_k} = 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left(\frac{\partial y(x_i;\mathbf{a})}{\partial a_j} \frac{\partial y(x_i;\mathbf{a})}{\partial a_k} - [y_i - y(x_i;\mathbf{a})] \frac{\partial^2 y(x_i;\mathbf{a})}{\partial a_j \partial a_k} \right)$$

$$\approx 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \frac{\partial y(x_i;\mathbf{a})}{\partial a_j} \frac{\partial y(x_i;\mathbf{a})}{\partial a_k} \tag{B.4}$$

The term $[y_i - y(x_i;a)]$ should reflect the (hopefully) small random measurement error at each data point, and as such should not contribute significantly to the summation for good models and reasonable data. The effect of neglecting the second derivative term is simply to alter the search path taken along the hypersurface to the χ^2 minimum - the value of the minimum itself is unaltered. A further advantage in ignoring the last term is that the routine is made more robust to outlier points.

It is usual to define from equations (B.3) and (B.4) the quantities

$$\beta_j \equiv -\frac{1}{2}\frac{\partial\chi^2}{\partial a_j} \tag{B.5}$$

$$\alpha_{jk} \equiv \frac{1}{2}\frac{\partial^2\chi^2}{\partial a_j \partial a_k} \tag{B.6}$$

The symmetric matrix α is called the curvature matrix since it describes the curvature of the χ^2 hypersurface in parameter space.

Having defined the function to be minimised, it is now necessary to develop a search strategy to optimally update the initial guess for the parameters **a**.

B.3 Finding the χ^2 minimum

By definition, the gradient $\nabla\chi^2$, or that direction in which χ^2 increases most rapidly, is a vector whose components are equal to the rate at which χ^2 increases in that direction. The *steepest descent* search algorithm is based simply upon taking small steps in the opposite direction to $\nabla\chi^2$ in parameter space, i.e.

$$\mathbf{a}_{new} = \mathbf{a}_{current} - \frac{\gamma}{2}\nabla\chi^2(\mathbf{a}_{current}) \tag{B.7}$$

where $\mathbf{a}_{new}$ is the new vector of parameters, $\mathbf{a}_{current}$ is the vector of current parameter values and $\frac{\gamma}{2}$ is some small constant which determines the step size. Defining the parameter update vector

$$\delta\mathbf{a} = \mathbf{a}_{new} - \mathbf{a}_{current} \tag{B.8}$$

then equation (B.7) can be written simply as

$$\delta\mathbf{a} = \gamma\beta \tag{B.9}$$

where the vector β is defined by equation (4.5).

It is well known however that the steepest descent method is only useful when the initial parameter vector **a** is far from the optimal parameters $\mathbf{a}_{opt}$ that minimise χ^2.

When $\mathbf{a} \approx \mathbf{a}_{opt}$, the steepest descent method has poor convergence properties. A preferable technique is to assume that near the χ^2 minimum the hypersurface is approximately parabolic with respect to a variation in the parameters **a**. This is equivalent to taking the first three terms of a Taylor's series expansion about some origin $\bar{\mathbf{a}}$ near the minimum. Any other point nearby in parameter space can then be written as

$$\chi^2(\mathbf{a}) = \chi^2(\bar{\mathbf{a}}) + \sum_j \frac{\partial \chi^2}{\partial a_j} a_j + \frac{1}{2} \sum_{jk} \frac{\partial^2 \chi^2}{\partial a_j \partial a_k} a_j a_k + \ldots$$

$$= \chi^2(\bar{\mathbf{a}}) - \mathbf{b} \bullet \mathbf{a} + \frac{1}{2} \mathbf{a} \bullet \mathbf{A} \bullet \mathbf{a} \qquad \text{(B.10)}$$

where $\mathbf{b} = -\nabla \chi^2 \big|_{\bar{\mathbf{a}}}$ and $A_{jk} = \frac{\partial^2 \chi^2}{\partial a_j \partial a_k} \big|_{\bar{\mathbf{a}}}$.

Taking the gradient of equation (B.10) with respect to the parameters a_j yields

$$\nabla \chi^2(\mathbf{a}) = \mathbf{A} \bullet \mathbf{a} - \mathbf{b} \qquad \text{(B.11)}$$

If **a** represents the optimal set of parameters, then $\nabla \chi^2 = 0$ and

$$\mathbf{A} \bullet \mathbf{a}_{opt} = \mathbf{b} \qquad \text{(B.12)}$$

Subtracting equation (B.11) from equation (B.12) and re-arranging gives the strategy for determining the optimal parameters from the initial vector as

$$\mathbf{a}_{opt} - \mathbf{a} = \mathbf{A}^{-1}[\nabla \chi^2(\mathbf{a})] \qquad \text{(B.13)}$$

This technique is known as *parabolic extrapolation*. In order to compare this result with the steepest descent algorithm, equation (B.13) is best expressed in terms of the parameter update vector $\delta \mathbf{a}$, the vector β and the matrix α as

$$\alpha . \bullet \delta \mathbf{a} = \mathbf{b} \qquad \text{(B.14)}$$

or

$$\delta a_j = \sum_k \varepsilon_{jk} \beta_k \qquad \text{(B.15)}$$

where the matrix ε is the inverse of α.

A disadvantage of using the parabolic extrapolation technique is that it cannot be relied upon to approach the minimum from any region where the χ^2 hypersurface is not approximately parabolic. Specifically, regions where the curvature is negative will lead to solutions of equation (B.15) that are unreliable.

The choice of using either the steepest descent or the parabolic extrapolation technique is clearly dependent upon the accuracy with which the initial conditions can be specified, and the quality of the experimental data. If the initial values of the parameters are not well known, then the steepest descent algorithm is the more robust method of finding the χ^2 minimum. Once the minimum is approached however, use of the parabolic extrapolation technique is preferred.

B.4 The Levenberg-Marquardt algorithm

The *Levenberg-Marquardt* (or *gradient expansion*) algorithm combines the best features of both steepest descent and parabolic extrapolation techniques. Recalling the parameter update formulae for each method

$$\frac{1}{\gamma}\delta a = \beta \qquad \text{(steepest descent)}$$

$$\alpha \bullet \delta a = \beta \qquad \text{(parabolic extrapolation)}$$

then a new matrix $\tilde{\alpha}$ is constructed by defining

$$\tilde{\alpha}_{jj} \equiv \alpha_{jj}(1+\lambda) \qquad \text{(B.16)}$$

$$\tilde{\alpha}_{jk} \equiv \alpha_{jk} \quad ; \quad j \neq k \qquad \text{(B.17)}$$

where λ is a scalar quantity known as the *Marquardt parameter*. The new parameter update formula now takes the form

$$\tilde{\alpha} \bullet \delta a = \beta \qquad \text{(B.18)}$$

When λ is very small or zero, equation (B.18) is just the parabolic extrapolation technique. However, when λ is large compared to the $\tilde{\alpha}_{jk}$, the matrix $\tilde{\alpha}$ is forced to be diagonally dominant, and equation (B.18) becomes

$$\tilde{\alpha}_{jj}\lambda\delta a = \beta \qquad \text{(B.19)}$$

which is just the steepest descent algorithm with $\tilde{\alpha}_{jj}\lambda \Leftrightarrow \frac{1}{\gamma}$. Thus, by adjusting the size of λ, the technique can be forced into either a steepest descent or a parabolic search of the χ^2 hypersurface. The formal solution for the parameter update vector is then

$$\delta a = \beta \bullet \mathfrak{E} \qquad \text{(B.20)}$$

where $\mathfrak{E}$ is the inverse of the pseudo-curvature matrix $\tilde{\alpha}$.

The Levenberg-Marquardt algorithm proceeds in the following manner:

(1) calculate $\chi^2(\mathbf{a})$

(2) start with a small value of λ, typically 0.001

(3) solve equation (B.20) for $\delta\mathbf{a}$ and evaluate $\chi^2(\mathbf{a}+\delta\mathbf{a})$

(4) if $\chi^2(\mathbf{a}+\delta\mathbf{a}) \geq \chi^2(\mathbf{a})$ increase λ by a substantial factor (typically 10) and go to step (3)

(5) if $\chi^2(\mathbf{a}+\delta\mathbf{a}) < \chi^2(\mathbf{a})$ decrease λ by a substantial factor (typically 10), update the trial solution $\mathbf{a}+\delta\mathbf{a} \rightarrow \mathbf{a}$, and go to step (3)

The routine can be terminated when all parameters have sufficiently converged, when χ^2 does not significantly decrease or after a maximum number of iterations. A further check for good convergence is that the Marquardt parameter be small compared to the α_{jj}.

B.5 Error estimation in non-linear least-squares

Since the result of the least-squares minimisation of a non-linear function requires some form of search routine rather than an exact analytic solution, there is no analytical form for the estimated error in each of the fit parameters a_j. The best that can be achieved is to develop an algorithm which is reasonable and reduces to the correct expression for a function that is linear in **a**.

In order to achieve this, it is appropriate to briefly review the derivation of the exact analytic expression for the estimated errors as they occur in the case of linear least-squares. In this case the optimal parameters **a** are determined from the linear equivalent of equation (B.15), which is

$$a_j = \sum_k \varepsilon_{jk} \beta_k \qquad \text{(B.21)}$$

where the model is

$$y(x_i) = \sum_j a_j b_j(x_i) \qquad \text{(B.22)}$$

with

$$\beta_k \equiv -\frac{1}{2}\frac{\partial \chi^2}{\partial a_k} = \sum_{i=1}^{N} \frac{y_i b_k(x_i)}{\sigma_i^2} \qquad \text{(B.23a)}$$

and

$$\varepsilon = \alpha^{-1} \text{ with } \alpha_{jk} \equiv \frac{1}{2}\frac{\partial^2 \chi^2}{\partial a_j \partial a_k}$$

$$= \sum_{i=1}^{N} \frac{1}{\sigma_i^2} b_j(x_i)\, b_k(x_i) \qquad \text{(B.23b)}$$

Now, the standard deviation σ_{a_j} representing the uncertainty in the determination of any parameter a_j is the square root of the sum of the squares of the product of the standard deviation of each data point σ_i multiplied by the effect that data point has on the determination of parameter a_j[1], i.e.

$$\sigma_{a_j}^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i}\right)^2 \qquad \text{(B.24)}$$

From equation (B.21)

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{M} \frac{\partial \beta_k}{\partial y_i} \varepsilon_{jk} \qquad \text{(B.25)}$$

since, from equation (B.23b), α_{jk} is independent of y_i, and so $\varepsilon = \alpha^{-1}$ is independent of y_i also. From equation (B.23a)

$$\frac{\partial \beta_k}{\partial y_i} = \frac{1}{\sigma_i^2} b_k(x_i) \qquad \text{(B.26)}$$

and so

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{M} \frac{1}{\sigma_i^2} b_k \varepsilon_{jk} \qquad \text{(B.27)}$$

Substituting equation (B.27) into equation (B.24) yields

$$\sigma_{a_j}^2 = \sum_{k=1}^{M} \sum_{r=1}^{M} \varepsilon_{jk} \varepsilon_{jr} \left(\sum_{i=1}^{N} \frac{1}{\sigma_i^2} b_k(x_i)\, b_r(x_i) \right)$$

$$= \sum_{k=1}^{M} \sum_{r=1}^{M} \varepsilon_{jk} \varepsilon_{jr} \alpha_{kr} \qquad \text{(by equation (B.23b))}$$

$$= \sum_{k=1}^{M} \varepsilon_{jk} \left(\sum_{r=1}^{M} \varepsilon_{jr} \alpha_{kr} \right)$$

$$= \sum_{k=1}^{M} \varepsilon_{jk} \delta_{jk}$$

i.e.

$$\sigma_{a_j}^2 = \varepsilon_{jj} \qquad \text{(B.28)}$$

Thus, in the linear case, the diagonal elements of the inverse curvature matrix are the variances of the fitted parameters **a**. The off diagonal elements ε_{jk} ($k \neq j$) describe the covariances between the parameters a_j and a_k. The matrix ε is therefore known as the *covariance matrix* or *error matrix*..

Comparison of the linear and non-linear solutions (equations (B.21) and (B.15)) shows that the fundamental mathematical difference between the two problems is that in the non-linear case the parameter *increments* are calculated, rather than the parameters themselves. The error in the linear case is associated solely with the curvature matrix α, and from equations (B.23b) and (B.6) it is apparent that the definition of α is identical for both problems.

The approximation made in the non-linear case is that the curvature matrix calculated at the χ^2 minimum *still* reflects the variances of each of the fitted parameters, and so the final step in the Levenberg-Marquardt routine, once satisfactory convergence is achieved, is to set the Marquardt parameter λ equal to zero and to re-calculate α and then ε. The diagonal elements ε_{jj} are taken to be the estimates of the variance for each parameter a_j.

Equation (B.28) assumes that the variance σ_i^2 of each data point is known. If they are not known (i.e. $\sigma_i^2 = 1$), they must be estimated from the experimental data set using

$$\sigma_i^2 \approx s^2 = \frac{1}{N-M} \sum_{i=1}^{N} [y_i - y(x_i;\mathbf{a})]^2$$

$$= \frac{1}{N-M} \chi^2_{(1)} \qquad \text{(B.29)}$$

where $\chi^2_{(1)}$ denotes the sum of the squares of the residuals calculated with $\sigma_i = 1$. The variance of the parameter a_j is then calculated from

$$\sigma^2_{a_j} \approx s^2 \varepsilon_{jj}(\sigma_i=1) = \frac{\chi^2_{(1)}}{N-M} \varepsilon_{jj}(\sigma_i=1) \qquad \text{(B.30)}$$

The 95% confidence interval for each parameter is then just

$$\delta aj = 2\sigma_{a_j} \qquad \text{(B.31)}$$

irrespective of whether equation (B.28) or equation (B.30) is used to calculate σ_{a_j}.

APPENDIX C

THE README AND spg_fit HEADER FILES

```
/*
                SPG-Fit
                Copyright DSTO, 1990
                Authors: Shaun Troedson   Tony Lindsay
                SP Group
                Communications Division
                DSTO Salisbury
*/
```

This code may be copied, modified etc provided the above copyright and authors' names remain above the major routines. Any bugs or suggestions may be sent to the above address or try email on sct@dstos3.dsto.oz

Note that the source files should be placed in a general access read write directory since there will be frequent recompiling by the user.

Setting up the environment:

To set up the environment do the following:

1. Create a source file containing the model function or just copy from the functions provided in this directory. This is not essential.

2. Optionally include in the same file the 1st derivatives of this model w.r.t the parameters. This will improve speed and reduce round off errors.

3. Create a text file (fn_info) with the parameter names. The provided functions have info files in this directory with the prefix (info_).

4. Recompile using a simple make command.

Step 1,2:

The user model may be written in Pascal or C. The filenames are "cfn.c" or "pfn.p" for the C or Pascal routine respectively. The main procedure should be called fn(N,X,P,Y,deriv,dY) where

N is the number of data points;
X is an array of size NPTS (see spg_fit.h) of real (double) that contains the independent variable values. If the user fn is not in closed form, ie it is not possible to simply plug these values in to get the function value at that point, care should be taken that the values for the dependent variable match the corresponding values for X;
P is an array of size NPARAMS (see spg_fit.h) of real (double) that contains the values of the parameters;
Y is an array of size NPTS of real (double) that contains the theoretical curve dependent variable values and is set by the user program;
deriv is a flag which is only optionally used by the user program to determine whether to perform analytic derivative calculations. spg_fit calls the user routine to obtain values to plot and does not require the derivative information, hence in order to save time the user should use this parameter to conditionally perform the derivative calculation when this flag is 1.
dY is a matrix of size [NPARAMS,NPTS] of real (double) returned by the user program. Each row contains the derivatives of the of the function w.r.t each parameter over the values of the array X. This is optional and if spg_fit does not detect that the analytic derivatives

exist it will perform a numeric calculation. It should be noted however that this entails taking the difference of two similar numbers and hence may suffer due to round off error.

Step 3:

The file "fn_info" is a text file that contains the parameter names- line by line. Note that these should be in the same order as implied in the array P above. If the supplied routines are used then their corresponding info file can be used.

Step 4: Compiling.

The fitting routine is now ready for compiling. The file Makefile contains two variables which should be defined. RESIDE is the directory in which the source codes and object files for spg_fit live. DEST is the directory in which the user wishes to place the final executable. The file "spg_fit.h" contains other definitions which may be specified. Here the maximum number of points (NPTS) and maximum number of parameters (NPARAMS) are defined. Also defined are STEPSIZE (the step taken in calculating the numerical derivatives), CRITERION (the convergence criterion), and the local fonts library.

Finally a simple make command is used to compile and link. If you are in the source directory for spg_fit then type

```
make fitp  --or--  make fitc
```

if the user routine is in Pascal or C respectively. You may also use the -f option of make if you are not in the source directory. The routine may now be executed by typing

```
fit
```

The Data file:

The format of the text file consists of 2,3 or 4 columns. The first column is taken as the independent variable values and the second column the dependent variable data. The 3rd and 4th columns, which are optional, contain the variances of the individual data points. These columns are used when performing a weighted fit,though the 4th column is only used when performing a X & Y weighted linear fit. The 3rd column is taken to be the variances in the independent column, or, if the 4th column does not exist, the variances in the dependent column. The columns may be separated by spaces or tabs. Note that the data should be presorted otherwise the results are unpredictable.

Some notes on operation:

The routine allows parameters to be changed on the screen. However it only updates the chi squared value if the number of samples is equal to the number of data points (this is the default). This is so that two calls to the function are not required which may be annoying for slow functions.

The routine uses a Levenberg-Marquardt routine for the non-linear fitting, hence the field for lambda. The linear fitting uses a York routine to do a X & Y weighted linear fit. References may be found in the documentation.

Mar 8 10:03 1991 spg_fit.h Page 1

```
/** NPTS: maximum number of allowed data and plot points
        hopefully this will be malloc'd in a later version*/
#define NPTS    250

/** NPARAMS: maximum number of allowed paramters*/
#define NPARAMS 10

/** STEPSIZE: this is the divisor of a parameter in calculating
        the derivatives of chi sq wrt this parameter. This
        constant may be optimized for the application*/
#define STEPSIZE        1e6

/**CRITERION: the convergence criterion, fitting will stop if the
        fractional change  of all parameters between one iteration
        and the next is less than this number*/
#define CRITERION       1e-5

/** NSD: number of standard deviations to use for errors*/
#define       NSD     2


/**MAX_ITERATIONS: if the LM routine gets into a runaway process
        it will stop at this value*/
#define MAX_ITERATIONS  50

/**FONTS: this defines your font directory. BOLD_FONT will be the
        font used for field labels. PLAIN_FONT will be the
        font used for field values.*/
#define BOLD_FONT       "/usr/lib/fonts/fixedwidthfonts/cour.b.12"
#define PLAIN_FONT      "/usr/lib/fonts/fixedwidthfonts/cour.r.12"
```

APPENDIX D

A LISTING OF SUPPLIED FUNCTIONS INCLUDING PASCAL AND C ROUTINES

```
Mar  8 10:34 1991  fit_gauss_area.p Page 1


/*
        gaussian distribution function
        total area normalized to P[4]
        FWHM = P[2]
        centred at P[1]
        baseline of P[3];
*/

const   NPTS=250;               /*take care that these values are the same as*/
        NPARAMS=10;             /*in the spg_fit.h file*/
        fourln2=2.7725887;
        sqrtpi=1.772453851;

type
        parray=array[1..NPARAMS] of real;       /*parameter type array*/
        darray=array[1..NPTS] of real;          /*data points type array*/
        derivarray=array[1..NPARAMS,1..NPTS] of real;   /*derivative type array*/

procedure fn(N:integer;                 /*number of data points*/
                var x:darray;           /*independent values*/
                var P:parray;           /*parameter values*/
                var F:darray;           /*dependent values (returned)*/
                deriv:integer;          /*flag for derivatives to be performed*/
                var dF:derivarray);     /*derivative values (returned)*/

var
        i:integer;
        temp:darray;
        u:real;                 /*mean*/
        FWHM:real;              /*full width half maximum*/
        base:real;              /*offset from zero*/
        area:real;              /*area under curve*/

begin

        u:=P[1];
        FWHM:=P[2];
        base:=P[3];
        area:=P[4];

        for i:=1 to N do        /*evaluate function*/
        begin
                temp[i]:=sqrt(fourln2)*exp(-fourln2*sqr((x[i]-u)/FWHM))/sqrtpi;
                F[i]:=base+area*(temp[i]/FWHM);
        end;

        if deriv=1 then                 /*optional*/
                for i:=1 to N do        /*evaluate derivative w.r.t parameters*/
                begin
                        dF[1][i]:=2*fourln2*(x[i]-u)*area*temp[i]/(FWHM*sqr(FWHM));
                        dF[2][i]:=(2*fourln2*sqr((x[i]-u)/FWHM)-1)*area*temp[i]/sqr(
                        dF[3][i]:=1.0;
                        dF[4][i]:=temp[i]/FWHM;
                end;

end;
```

Mar 8 10:29 1991 fit_lorentz_area.c Page 1

```
/*
        lorentzian distribution function
        total area under curve normalized to P[3]
        P[0]=FWHM;
        centred at P[1]
        baseline P[2];
*/

#include <math.h>
#include "spg_fit.h"            /*spg_fit header file*/

fn(n,X,P,f,deriv,df)
int     n;                      /*number of data points*/
double  X[];                    /*independent values*/
double  P[];                    /*parameter values*/
double  f[];                    /*dependent values (returned)*/
int     deriv;                  /*flag for derivatives to be performed*/
double  df[][NPTS];             /*derivative values (returned)*/
{
        int i;
        double  temp[NPTS];
        double  Pi=3.1415926538;

        for (i=0;i<n;i++)       /*evaluate function*/
        {
                temp[i]=4.0*(X[i]-P[1])*(X[i]-P[1])+P[0]*P[0];
                f[i]=P[2]+2.0*P[3]*P[0]/(Pi*temp[i]);
        }

        if (deriv)              /*optional*/
                for (i=0;i<n;i++)       /*evalaute derivatives w.r.t params*/
                {
                        df[0][i]=(1.0-2.0*P[0]*P[0]/temp[i])*2.0*P[3]/(Pi*temp[i]);
                        df[1][i]=16*P[3]*P[0]*(X[i]-P[1])/(Pi*temp[i]*temp[i]);
                        df[2][i]=1.0;
                        df[3][i]=2.0*P[0]/(Pi*temp[i]);
                }
}
```

Mar 8 10:31 1991 fit_exp.c Page 1

```
/*
        exponential function:  y[i] = offset + Y[0]*exp(exponent*x[i]);
*/

#include <math.h>
#include "spg_fit.h"

fn(n,X,P,f,deriv,df)
int     n;                      /*number of data points*/
double  X[];                    /*independent values*/
double  P[];                    /*parameter values*/
double  f[];                    /*dependent values (returned)*/
int     deriv;                  /*flag for derivatives to be performed*/
double  df[][NPTS];             /*derivative values (returned)*/

{
        int i;

        for (i=0;i<n;i++)       /*evaluate function*/
                f[i]=P[2]+P[0]*exp(P[1]*X[i]);

        if (deriv)              /*optional*/
                for (i=0;i<n;i++)       /*evaluate derivs w.r.t params*/
                {
                        df[0][i]=exp(P[1]*X[i]);
                        df[1][i]=P[0]*X[i]*exp(P[1]*X[i]);
                        df[2][i]=1.0;
                }
}
```

```
Mar  8 10:33 1991  fit_gauss_ht.p Page 1


/*
        gaussian distribution function
        normalized to height P[4]
        P[2] is the FWHM
        centred at P[1]
        baseline P[3]
*/

const   NPTS=200;       /*take care that these values are the same as*/
        NPARAMS=10;     /*in the spg_fit.h file*/
        fourln2=2.7725887;

type
        parray=array[1..NPARAMS] of real;       /*parameter type array*/
        darray=array[1..NPTS] of real;          /*data points type array*/
        derivarray=array[1..NPARAMS,1..NPTS] of real;   /*derivative type array*/

procedure fn(N:integer;                 /*number of data points*/
                var x:darray;           /*independent values*/
                var P:parray;           /*parameter values*/
                var F:darray;           /*dependent values (returned)*/
                deriv:integer;          /*flag for derivatives to be performed*/
                var dF:derivarray);     /*derivative values (returned)*/

var
        i:integer;
        temp:darray;
        u:real;
        FWHM:real;
        base:real;
        height:real;

begin

        u:=P[1];
        FWHM:=P[2];
        base:=P[3];
        height:=P[4];

        for i:=1 to N do                /*evalaute function*/
        begin
                temp[i]:=exp(-fourln2*sqr((x[i]-u)/FWHM));
                F[i]:=base+height*temp[i];
        end;

        if deriv=1 then                 /*optional*/
                for i:=1 to N do        /*evaluate derivs w.r.t params*/
                begin
                        dF[1][i]:=2*fourln2*(x[i]-u)*height*temp[i]/sqr(FWHM);
                        dF[2][i]:=2*fourln2*sqr(x[i]-u)*height*temp[i]/(FWHM*sqr(FWHI
                        dF[3][i]:=1.0;
                        dF[4][i]:=temp[i];
                end;

end;
```

Mar 8 10:35 1991 fit_lorentz_ht.c Page 1

```
/*
        lorentzian distribution function
        normalized to P[3] at line centre
        FWHM = P[0]
        centred at P[1] with baseline of height P[2]
*/

#include <math.h>
#include "spg_fit.h"

fn(n,X,P,f,deriv,df)
int     n;                      /*number of data points*/
double  X[];                    /*independent values*/
double  P[];                    /*parameter values*/
double  f[];                    /*dependent values (returned)*/
int     deriv;                  /*flag for derivatives to be performed*/
double  df[][NPTS];             /*derivative values (returned)*/

{
        int i;
        double  temp[NPTS];
        double  Pi=3.1415926538;

        for (i=0;i<n;i++)               /*evaluate function*/
        {
                temp[i]=4.0*(X[i]-P[1])*(X[i]-P[1])+P[0]*P[0];
                f[i]=P[2]+P[3]*P[0]*P[0]/temp[i];
        }

        if (deriv)                      /*optional*/
                for (i=0;i<n;i++)       /*evaluate derivs w.r.t params*/
                {
                        df[0][i]=(1.0-P[0]*P[0]/temp[i])*(2*P[3]*P[0]/temp[i]);
                        df[1][i]=8*(X[i]-P[1])*P[3]*P[0]*P[0]/(temp[i]*temp[i]);
                        df[2][i]=1.0;
                        df[3][i]=P[0]*P[0]/temp[i];
                }
}
```

Mar 8 10:37 1991 fit_power_fn.c Page 1

```
/*
        power function: y[i] = intercept+scale*x[i]^exponent
*/

#include <math.h>
#include "spg_fit.h"

fn(n,X,P,f,deriv,df)
int     n;                      /*number of data points*/
double  X[];                    /*independent values*/
double  P[];                    /*parameter values*/
double  f[];                    /*dependent values (returned)*/
int     deriv;                  /*flag for derivatives to be performed*/
double  df[][NPTS];             /*derivative values (returned)*/

{
int i;
double  powx[NPTS];
double  lnx[NPTS];

        for (i=0;i<n;i++)       /*evaluate function*/
        {
                lnx[i]=log(X[i]);
                powx[i]=exp(P[0]*lnx[i]);
                f[i]=P[2]+P[1]*powx[i];
        }

        if (deriv)              /*optional*/
        {
                for (i=0;i<n;i++)       /*evaluate derivs w.r.t params*/
                {
                        df[0][i]=P[1]*lnx[i]*powx[i];
                        df[1][i]=powx[i];
                        df[2][i]=1.0;
                }
        }
}
```

Mar 8 10:39 1991 fit_sin.c Page 1

```
/*
        general sinusoid: y[i] = intercept + scale * sin( k*x[i] + phase );
*/

#include <math.h>
#include "spg_fit.h"            /*spg_fit header file*/

fn(n,X,P,f,deriv,df)
int     n;                      /*number of data points*/
double  X[];                    /*independent values*/
double  P[];                    /*parameter values*/
double  f[];                    /*dependent values (returned)*/
int     deriv;                  /*flag for derivatives to be performed*/
double  df[][NPTS];             /*derivative values (returned)*/

{
        int i;

        for (i=0;i<n;i++)       /*evaluate function*/
                f[i]=P[2]+P[0]*sin(P[1]*X[i]+P[3]);

        if (deriv)              /*optional*/
        {
                for (i=0;i<n;i++)       /*evaluate derivs w.r.t params*/
                {
                        df[0][i]=sin(P[1]*X[i]+P[3]);
                        df[1][i]=P[0]*X[i]*cos(P[1]*X[i]+P[3]);
                        df[2][i]=1.0;
                        df[3][i]=P[0]*cos(P[1]*X[i]+P[3]);
                }
        }
}
```

Department of Defence

DOCUMENT CONTROL DATA SHEET

Page Classification	
UNCLASSIFIED	
Privacy Marking/Caveat	

1a. AR Number	1b. Establishment Number	2. Document Date	3. Task Number
AR-006-481	ERL-0544-GD	APR 91	DEF 89-210

4. Title

A GENERAL LEAST-SQUARES FITTING PACKAGE FOR THE SUN WORKSTATION

5. Security Classification

Document	Title	Abstract
U	U	U

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L)

6. No. of Pages	54
7. No. of Refs.	3

8. Author(s)

S.C. Troedson and A.C. Lindsay

9. Downgrading/Delimiting Instructions

10a. Corporate Author and Address

Electronics Research Laboratory
PO Box 1600
SALISBURY SA 5108

10b. Task Sponsor

11. Officer/Position responsible for

Security..... N/A

Downgrading..... N/A

Approval for Release..... DERL

12. Secondary Release of this Document

APPROVED FOR PUBLIC RELEASE

Any enquiries outside stated limitations should be referred through DSTIC, Defence Information Services, Department of Defence, Anzac Park West, Canberra, ACT 2600.

13a. Deliberate Announcement

No Limitation

13b. Casual Announcement (for citation in other documents)

[X] No Limitation

[ ] Ref. by Author & Doc No only

14. DEFTEST Descriptors

Least-squares Method
Man Computer Interface
Computer Programs
Software Engineering

15. DISCAT Subject Codes

1205

16. Abstract

This document describes a general least-squares fitting software package, featuring an easy to use and flexible user interface. It is implemented on a Sun workstation and designed for the public domain.

Page Classification
UNCLASSIFIED
Privacy Marking/Caveat

16. Abstract (CONT.)

17. Imprint

Electronics Research Laboratory
PO Box 1600
SALISBURY SA 5108

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
ERL-0544-GD	603855	GENERAL DOCUMENT

21. Computer Programs Used

C Language

22. Establishment File Reference(s)

23. Additional information (if required)